Assignment 1: CS7641 - Machine Learning

Yurong Fan

Feb 1, 2019

1. The Classification Problem and the Data Set

1.1 Forest Cover Type Classification

The problem is predicting forest cover type from features such as elevation, aspect, slope, distance to hydrology, distance to roadways, and soil type. There are 54 attributes (one-hot-coded categorical variables and numerical attributes), 581012 instance, and 7 classes in the original dataset. I simplified the problem by converting it into a binary classification. The two most rare types of cover type is combined into the positive class, and the rest classes are combined as the negative class. Let's imagine that forest protectors will use this classifier to find environment suitable for the rare cover types, so they can grow the rare cover types. Thus a good classifier to this problem should not favor the dominate cover types, but well predicts the rare cover types.

I further took a 10% stratified sample simply because the dataset is too big for some computing expensive algorithms on my local machine. To make below analysis still repeatable, a random seed (999) was set when taking the sample. (I can oversample the rare classes to improve prediction performance on those classes. However it is another topic and I would like to preserve the class distribution in this analysis.) The resulted data for modeling has below distribution.

Cover Type	Number of instances
1 – rare cover types	3850
0 – dominate cover types	52200

Cover type distribution

The original dataset can be downloaded from https://archive.ics.uci.edu/ml/datasets/Covertype.

1.2 Digit Recognition

The digit recognition is one of the most classic computer vision problems. The features in this dataset are pixel values of handwritten images of 10 digits.

I think it's interesting to compare performance of different algorithms on this dataset and the forest cover type dataset as they are very different in natural. More specifically, features in the cover type dataset are explicit human understandable features that are ready. The job of the algorithms is learning the relationship between those features and the targets. However the features in the digit recognition problem are quite "raw".

https://www.kaggle.com/c/digit-recognizer

2. Performance Evaluation

2.1 Evaluation Metric

Among different classifier performance metrics such as accuracy, AUC, confusion matrix or precision and recall, F1 score, I choose F1 score, more precisely, macro F1 score as the metric to evaluate the performance of different classifiers. The reasons are explained below.

1) The end goal of the two classification problems is to predict the class one instance belongs to rather than ranking instances' probabilities of belonging to one class, or estimating the probability of belonging to one class. Even though these three goals are all different faces of the classification problem and are highly related with each other, accuracy, confusion matrix or precision and recall, and F1 score can evaluate the performance to the first goal (our goal) directly, while AUC is a better metric for ranking problem.
2) The problem with accuracy (percentage of instances with the right classification) is that a) our problems have unbalanced class distributions, and accuracy has the tendency to favor classifiers who predict more dominate class; b) accuracy is oversimplified in the sense that it cannot evaluate the trade-off between true positive and true negative which is usually important in classification problems of the first goal. Confusion matrix or precision and recall can solve these problems.
3) F1 score combines precision and recall into a single metric and it simplifies comparison across algorithms.

2.2 Evaluation Method

Apart from using F1 score as the single metric for performance evaluation, cross validation is used to evaluate performance in both the training and the testing data for below reasons.

1) Cross validation can enable performance evaluation in both training and testing data. I used it when tuning model complexity related hyper parameters to monitor the bias and variance tradeoff, and select parameters that can give low bias and variance.
2) Though simple train and test split can also give performance evaluation in both training and testing data, cross validation can make full use of the data as all instance will be in the training fold.

A fold number of 5 was used in my cross validation so it is not too computational expensive (like a fold number of 10), and gives a reasonable split ratio (80%) between training and testing data.

2.3 Baseline Performance

In order to know how much performance is simply due to the data or the problem, and how much performance is improved by the classifier, the performance of a classifier that predict all instances belonging to the dominate class is set as the baseline performance.

3. Classification Algorithms and Performance Tuning

Most algorithms were implemented with scikit-learn version 0.20.2.

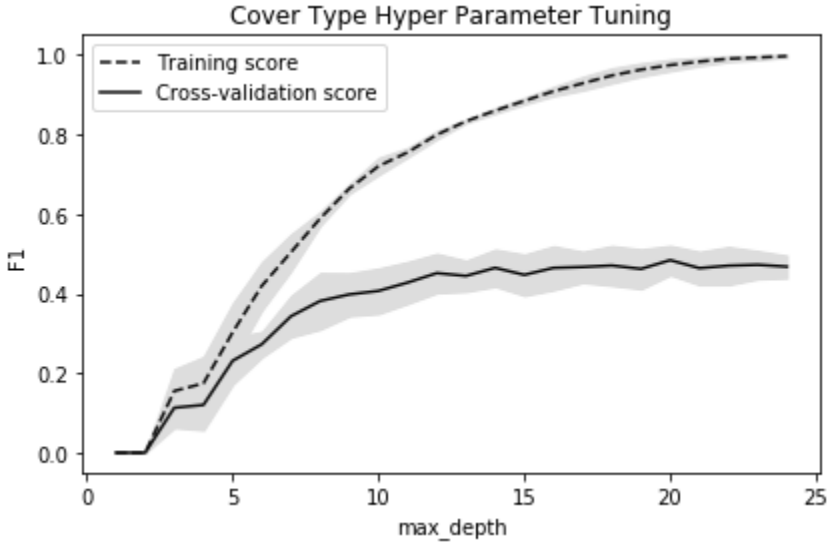
3.1 Decision Trees

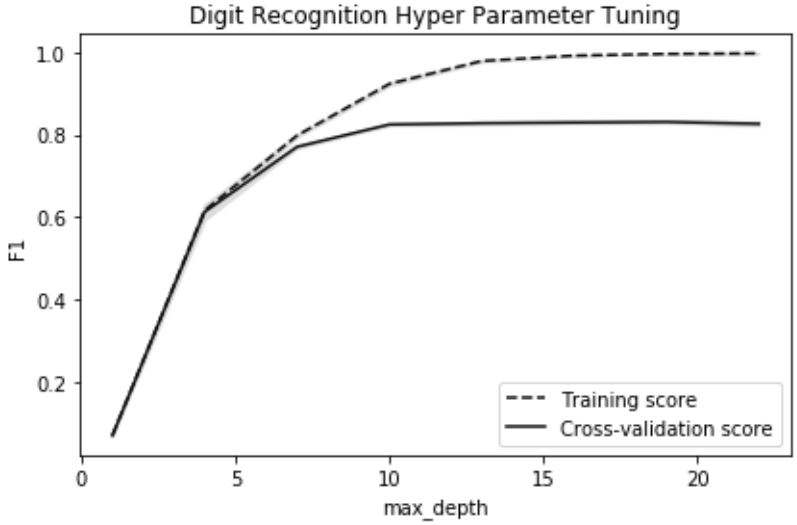
Decision tree was implemented with scikit-learn Decision Tree. It uses the CART (Classification and Regression Trees) algorithm that allows both categorical and numerical values as input, and constructs binary trees using the feature and threshold that yield the largest information gain at each node.

Without pruning, the decision tree fits both data perfectly with 1.0 F1 score in the training data, but only got 0.48/0.83 F1 score the testing data in the Cover Type and Digit Recognition dataset respectively. This is a clear sign of overfitting.

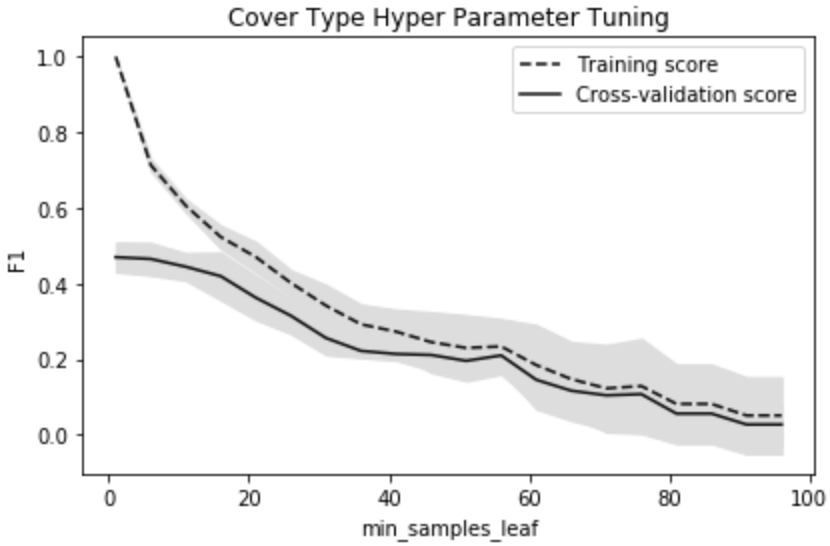
To perform regularization and prevent overfitting, I tested different pruning parameters such as max depth of the tree, minimum number of instances for a leaf.

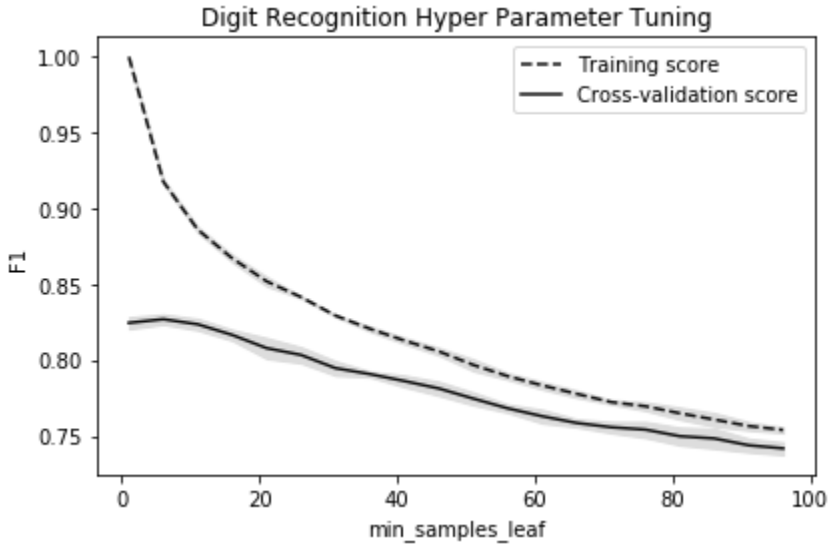
As the number of max depth increase, the training F1 scores keep increasing until it reached 1.0, while test F1 scores from cross validation stop increasing at around 10 or slightly decrease as the decision tree becomes too flexible and captures noise rather than signal.





For minimum samples in a leaf, the Cover Type dataset has the highest test F1 score when minimum samples in a leaf equals 1 and the Digit Recognition dataset has the highest test F1 score when minimum samples in a leaf equals 6. However 1 is still not a value to choose for the Cover Type dataset as the gap between training and testing score is huge. It is a sign of overfitting, and is not a good balance between bias and variance.



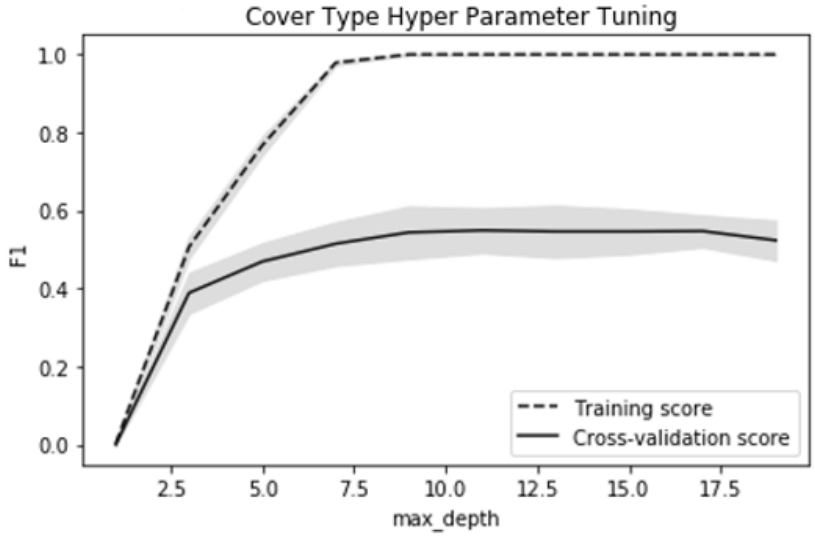


3.2 Gradient Boosting Trees

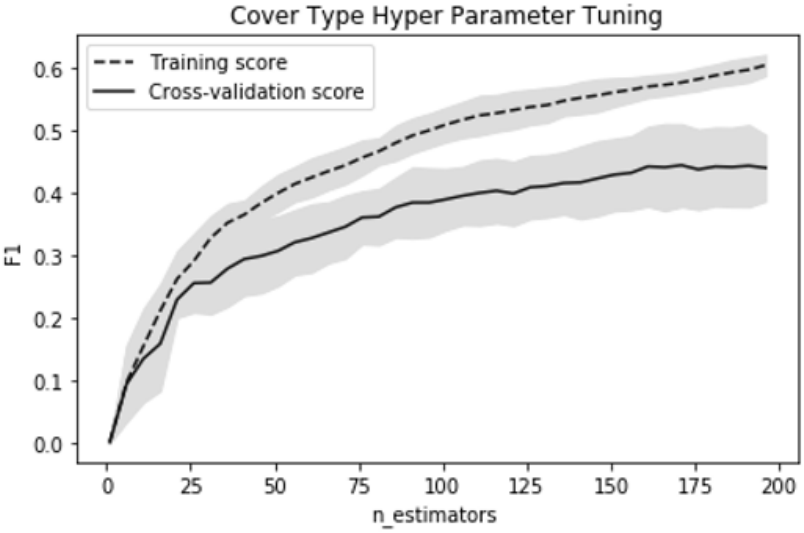
I implemented the boosting tree algorithm with [scikit-learn Gradient Boosting](). It's a generalization of AdaBoost which expand boosting tree beyond binary classification to support regression, and multi-class classification.

Same as Decision Tree, I turned the performance of Gradient Boosting Tree by pre-pruning the complexity (max depth of tree, minimum number of instances in leaves etc.) of base learner trees.

It is noteworthy that the performance on the test set stops increasing at earlier point with even higher F1 score comparing with the performance increase in the decision tree as the max depth increases. Boosting can combine weak learners into a strong learner is an explanation.

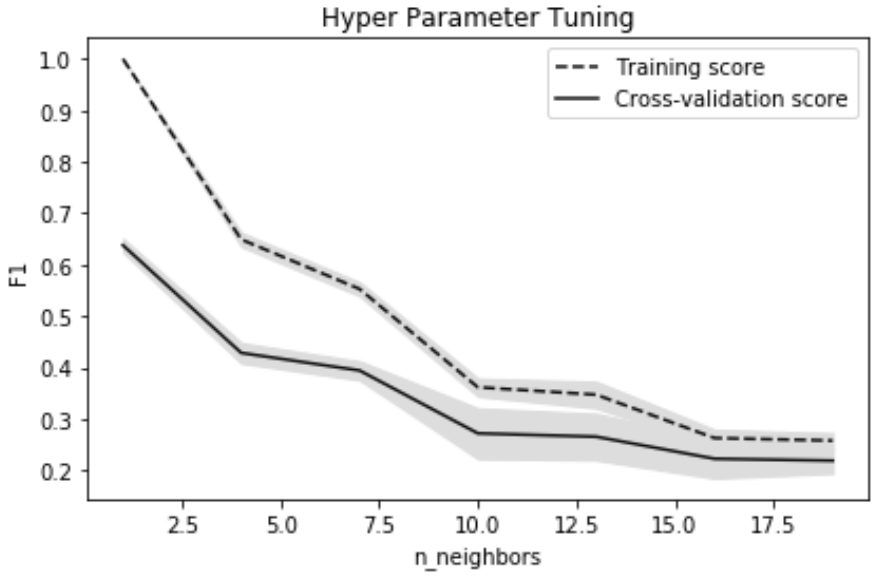


I also tested two regularization parameters that are only for gradient boosting, not for the decision tree, which are the number of base learner trees, and the learning rate. It's interesting to see that the gap between training score and testing score as I increase the number of learners (n_estimators) are not as large as the gap between I increase max depth



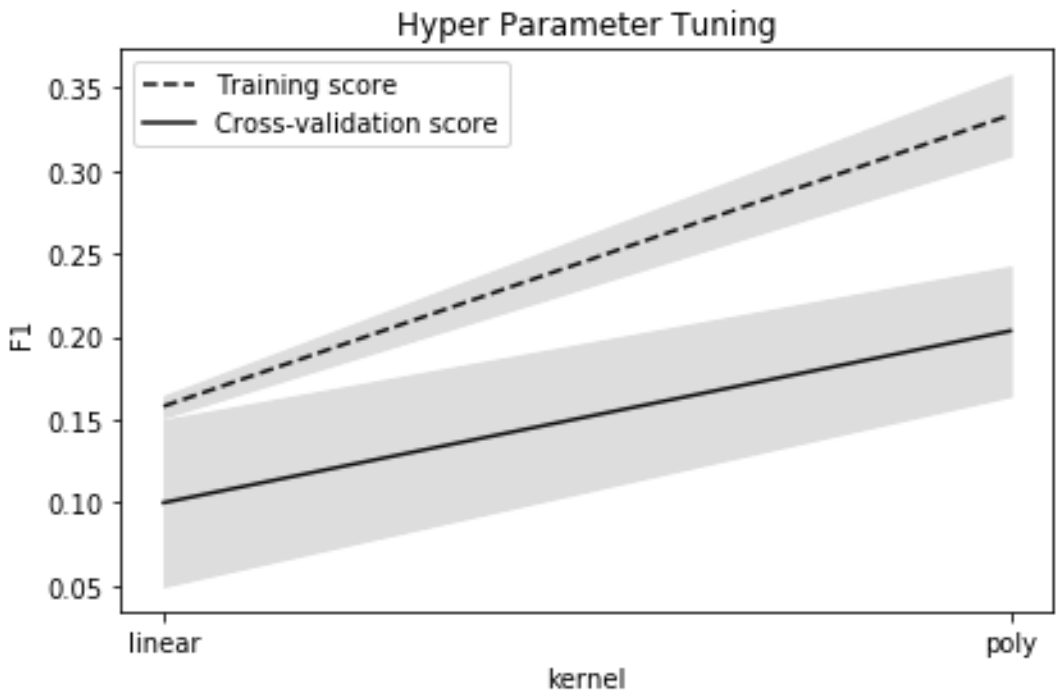
3.3 K-NN

Next, I implemented one lazy learner k-Nearest Neighbors with sklearn KNeighborsClassifier. To improve its performance, I tested different numbers of k. The Cover Type dataset favors small k. Also considering the best minimum number of instances in a leaf learned in Decision Tree, there might be local patterns present in the Cover Type dataset.



3.4 SVM

I implemented SVM with scikit-learn SVC. It allows different forms of kernel such as linear, polynomial, RBF, sigmoid to fit complex decision boundaries. The polynomial kernel function (with a degree of 2) performs better than the linear kernel indicating non-linear relationship between features and the target. But the training time of polynomial kernel increased dramatically.



3.5 Neural Network

I implemented neural network with [scikit-learn Neural network models] which uses Multi-layer Perceptron(MLP) algorithm. The performance was turned by performing a grid search of different values of number of hidden layers, and number of nodes in hidden layers, and change activation functions.

I first used the rectified linear unit function on the Cover Type dataset. In below setting of network structures, it reaches the best performance in (200, 100) (200 nodes in the first hidden layer, and 100 nodes in the second hidden layer). Further increase the number of nodes to (300, 150) actually worsen the performance.

Number of nodes in each layer	Mean Training F1	Mean Cross Validation F1
(6,)	0.03019788	0.04625582
(20,)	0.13120644	0.14634086
(40,)	0.12758463	0.12970584
(60,)	0.13084658	0.08491764
(100,)	0.17045021	0.15509485
(12, 6)	0.12937513	0.12710358
(20, 10)	0.11684061	0.12008564
(40, 20)	0.17905868	0.19257991
(60, 30)	0.23772226	0.23913071
(100,50)	0.31375243	0.28641041
(160, 80)	0.26502162	0.24808327

(200,100)	0.32272046	0.30751491
(300, 150)	0.24927177	0.22292837

Cover Type dataset - Neural Network tuning numbers of nodes and layers

Testing different activation function on the (200, 100) network structure, the rectified linear unit function does a better job than the sigmoid function on the Cover Type dataset.

Activation Function	Mean Training F1	Mean Cross Validation F1
The logistic sigmoid function	0.12172216	0.10943272
The rectified linear unit function	0.32272046	0.30751491

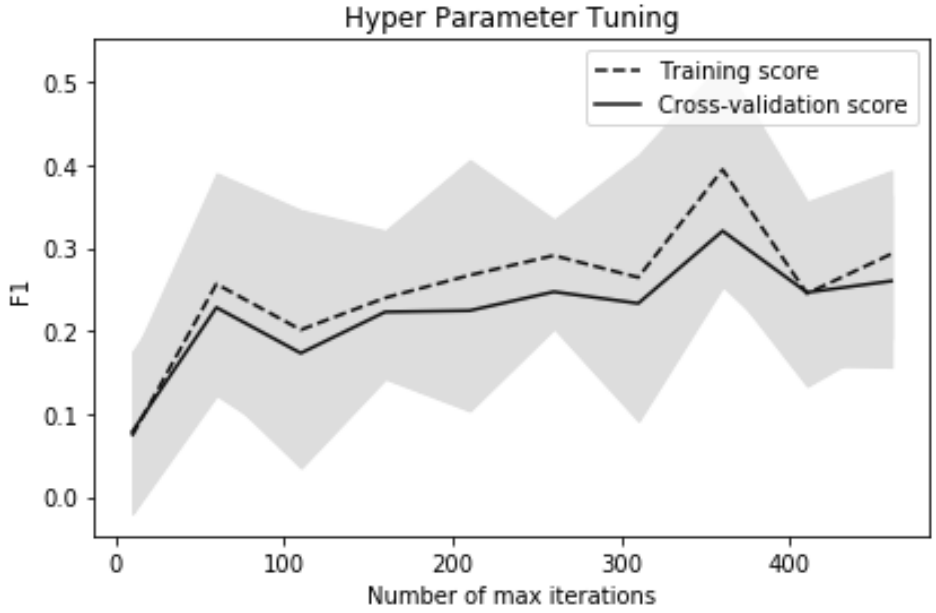
Cover Type dataset - Neural Network tuning activation function on (200, 100) network

For the Digit Recognition dataset, Neural Network achieves much higher performance comparing with the other algorithms.

Number of nodes in each layer	Mean Training F1	Mean Cross Validation F1
(100,)	0.99225667	0.93603235
(150,)	0.99182385	0.94358582
(60, 30)	0.9869368	0.93923036
(100, 50)	0.98034107	0.92939528

Digit Recognition dataset - Neural Network tuning numbers of nodes and layers

The number of iterations in the neural network is meaningful in saving computing effort and preventing overfitting. Below chart from Cover Type dataset shows a clear sign of “saturation” of performance as the number of iteration increases.



4. Final Comparison between Algorithms

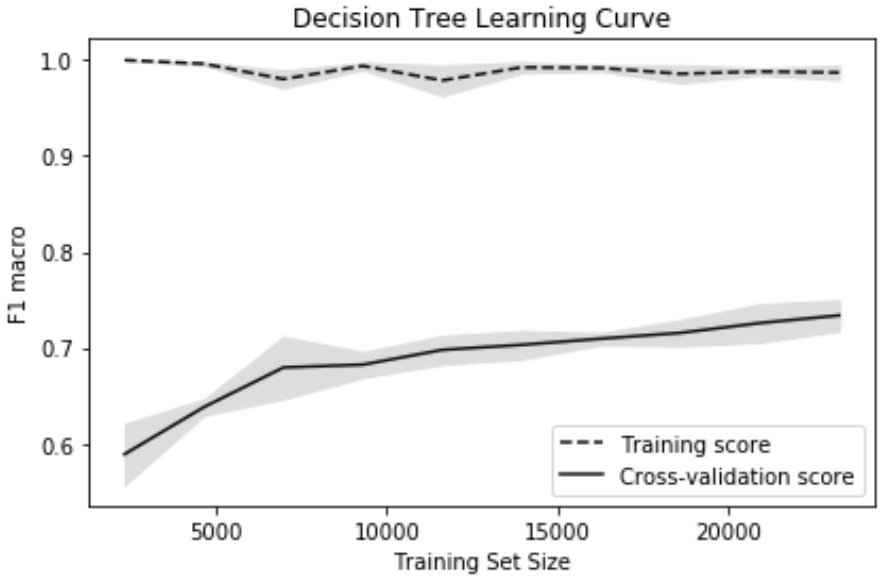
4.1 Best algorithm for each problem

For the cover type problem, K-NN with k = 3 and gradient boosting performs best as the dataset is relatively small (indicated from the learning curve) and the features are explicit.

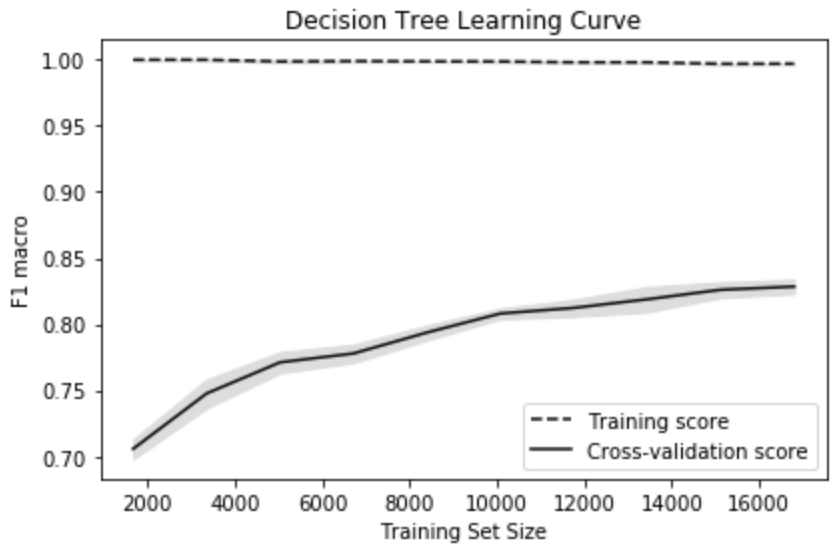
For the Digit recognition problem, neural network way out-perform other algorithms as there are no explicit features and the relationship between the dataset and the target are complex and non-linear.

4.1 Learning Curve

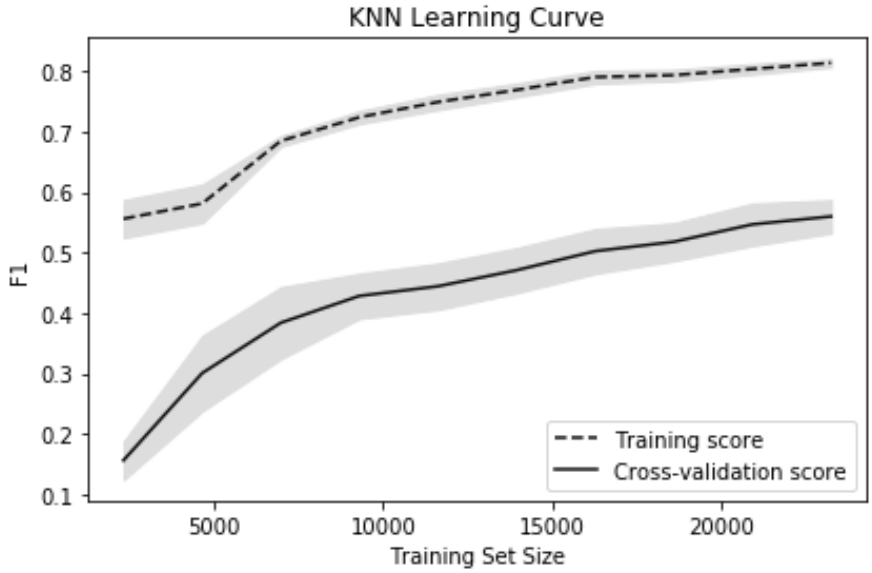
For all algorithms on both dataset, the performances are still in an increasing trend when reaching the full data size in the learning curves meaning all algorithms can benefit from further increasing the data size for both problems. But some performance of some algorithms are more stable as we increase the data size. For example, from below charts, performance of KNN climbs more quickly than performance of decision tree as data size increases.



Cover Type dataset - Decision Tree learning curve



Cover Type dataset - Decision Tree learning curve



Cover Type dataset – KNN learning curve

4.2 Learning Time

Kernel SVM has the longest learning time especially when using polynomial kernel. In terms of time complexity, it is the only algorithm whose training time increases in the quadratic order as the number of instances increases. The learning time of k-NN, Neural Network, and gradient boosting also depend the number of iterations, the number of neighbors, and the number of estimators, and the number of nodes in the neural network